Shaders em Jogos

Roberto Cezar Bianchini

Tópicos da Apresentação

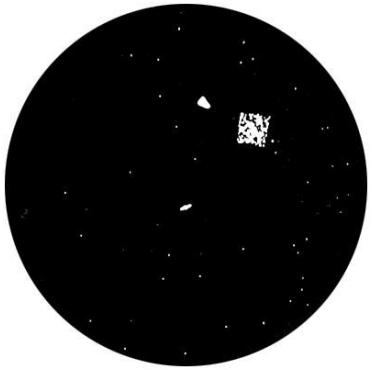
- Contextualização do tutorial.
- Pipeline Gráfico.
- Modelos de Iluminação.
- Shaders, Vertex, Geometry, Tessalation e Fragament Shaders.

Tópicos da Apresentação

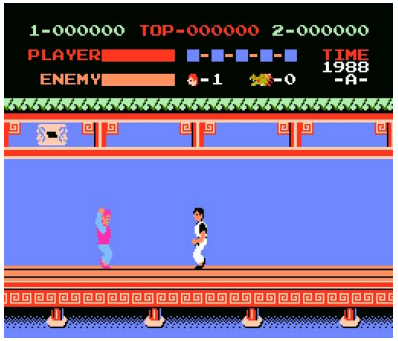
- **Contextualização do tutorial.**
- Pipeline Gráfico.
- Modelos de Iluminação.
- Shaders, Vertex, Geometry, Tessalation e Fragament Shaders.

Contextualização

Spacewar! (1961)



KungFu Master (1985)



Zelda OT (1998)

KOTOR (2003)

Fallout 4 (2015)

Aspectos Importantes

- A evolução gráfica dos jogos tende a aumentar cada vez mais.
- Será exigido dos desenvoveldores jogos com mais realismo gráfico, renderizados com um número de **Quadros Por Segundo** (*Frames Per Second – FPS*) que transmita a impressão de continuidade nas animações.
- Esta evolução será comprometida se apenas soluções do tipo *força-bruta*, que dependem totalmente no poder de processamento de hardwares cada vez mais avançados. **Novas técnicas deverão ser utilizadas em conjunto com hardwares mais potentes.**

Tópicos da Apresentação

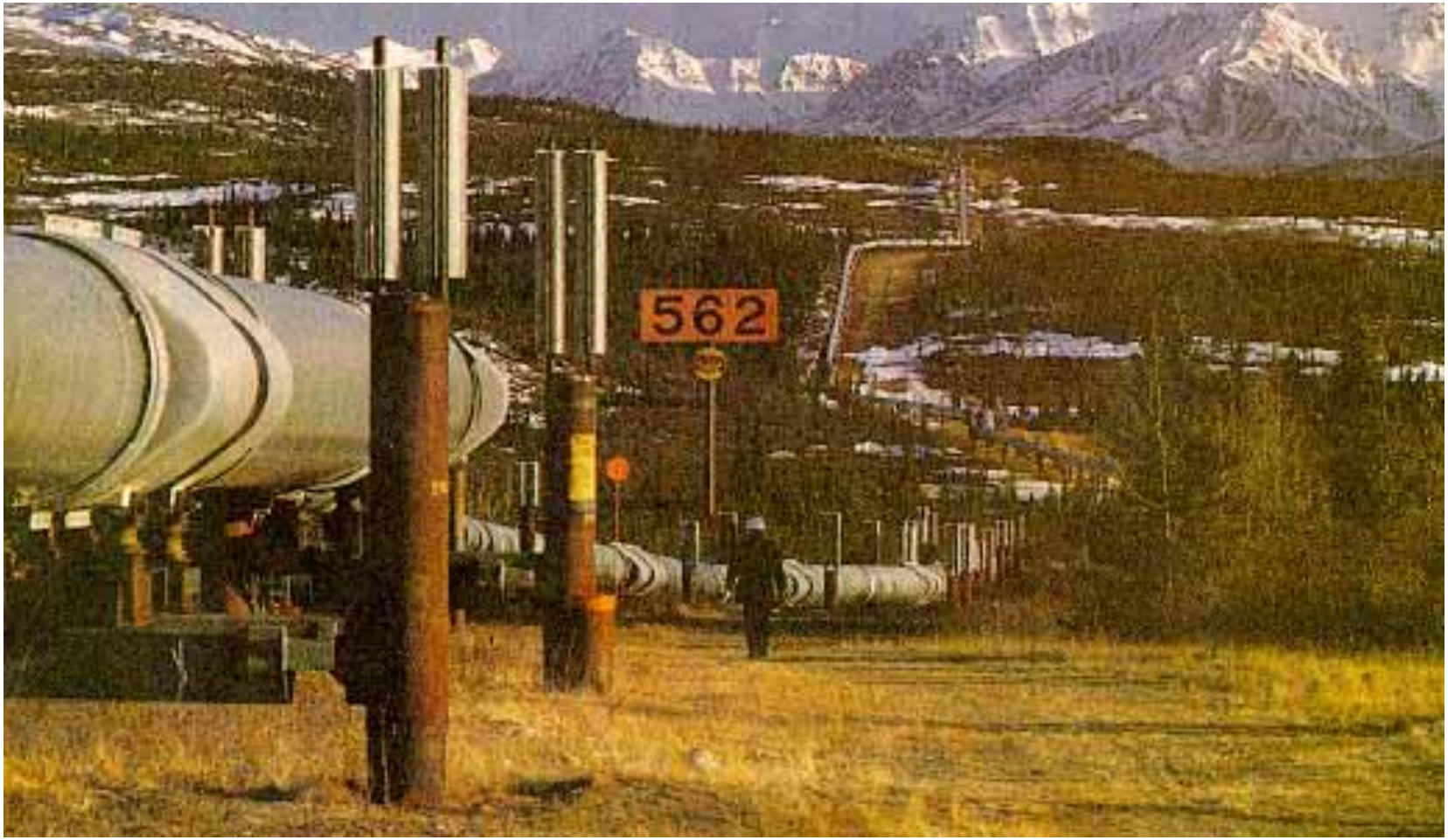
- Contextualização do tutorial.
- **Pipeline Gráfico.**
- Modelos de Iluminação.
- Shaders, Vertex, Geometry, Tesselation e Fragment Shaders.

Pipeline

- Estrutura com vários estágios, onde os elementos de um (ou vários) tipo(s) fluem entre estes, seqüencialmente, até atingir o fim da estrutura.
- Em cada estágio, operações podem ser efetuadas nos elementos vindos do estágio anterior e repassados para o próximo.
- Idealmente, uma tarefa de *n* estágios realizada sem a divisão em pipeline, tem um aumento de *n* processos completos por unidade de tempo quando dividida em pipeline.

Exemplos de Pipelines

Oleoduto



Exemplos de Pipelines

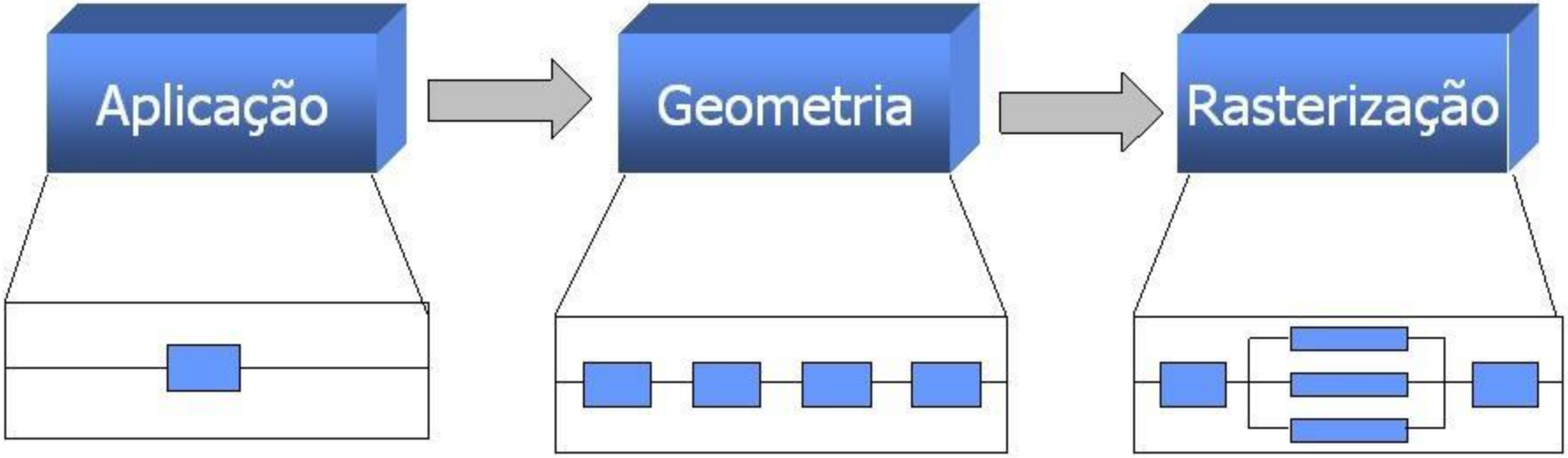
Indústria Automobilística

FÁBRICA DE MOTORES

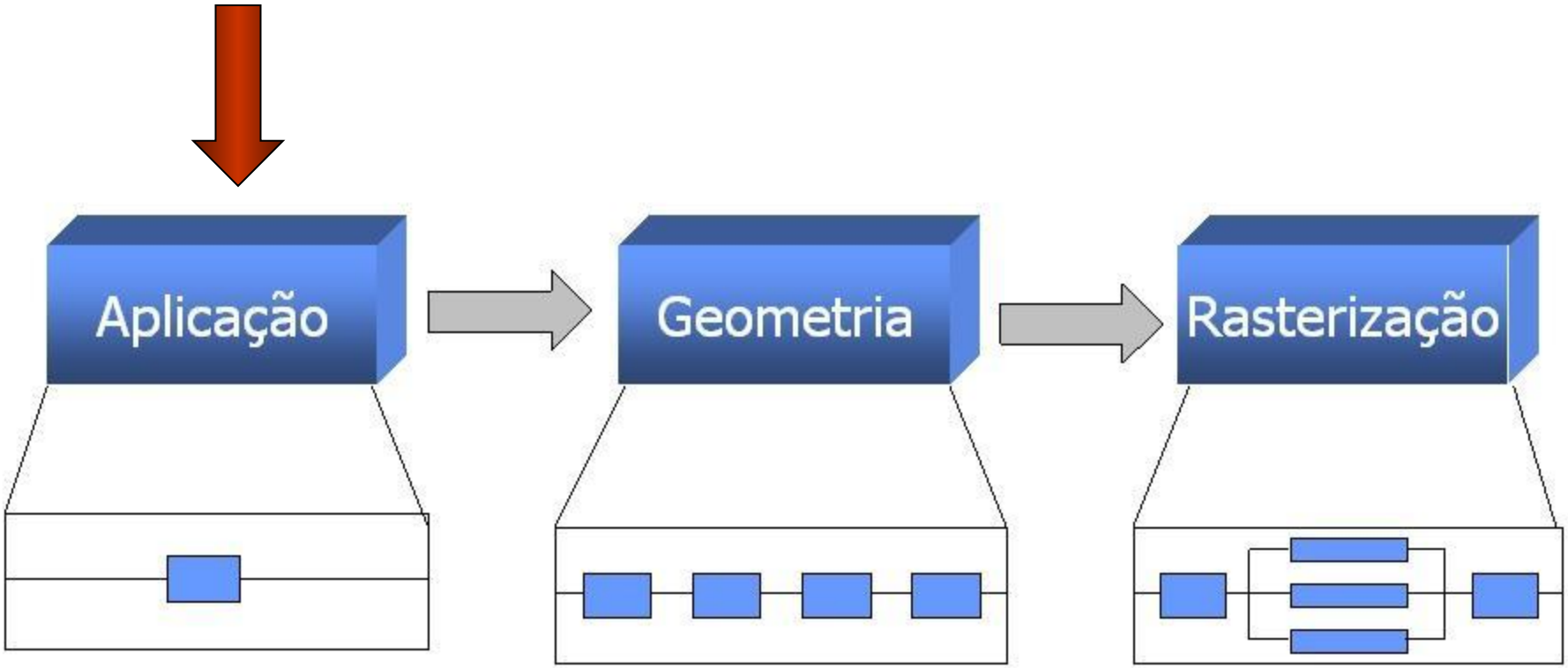
Pipeline Gráfico

- O processo de renderização de imagens também é do tipo pipeline.
- Os elementos de entrada que fluem pelos estágios são informações referentes aos objetos a serem exibidos – posição, cor, orientação – e a saída são imagens 2D – pixeis.

Pipeline Gráfico



Pipeline Gráfico

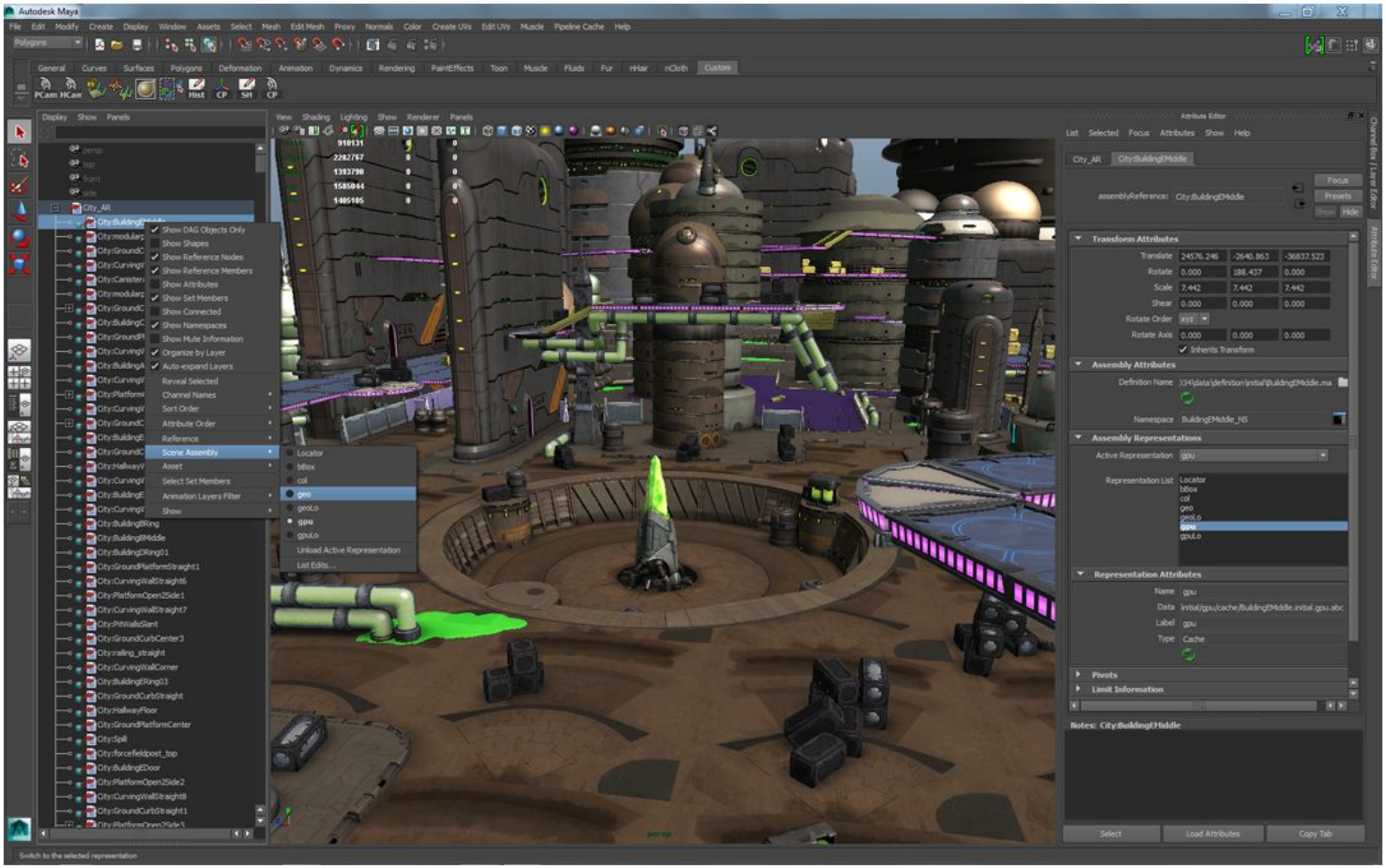


Aplicação

- **Aplicação** é o estágio do pipeline gráfico feita completamente em software, sobre a qual o desenvolvedor tem completo controle sobre o que fazer.
- Neste estágio se definem os objetos, suas características animações, entre outros.
- Operações típicas deste estágio são deteção de colisão, processamento da entrada do usuário, e outras otimizações.

Exemplos de Aplicações

Modelador



Maya 2014

Exemplos de Aplicações

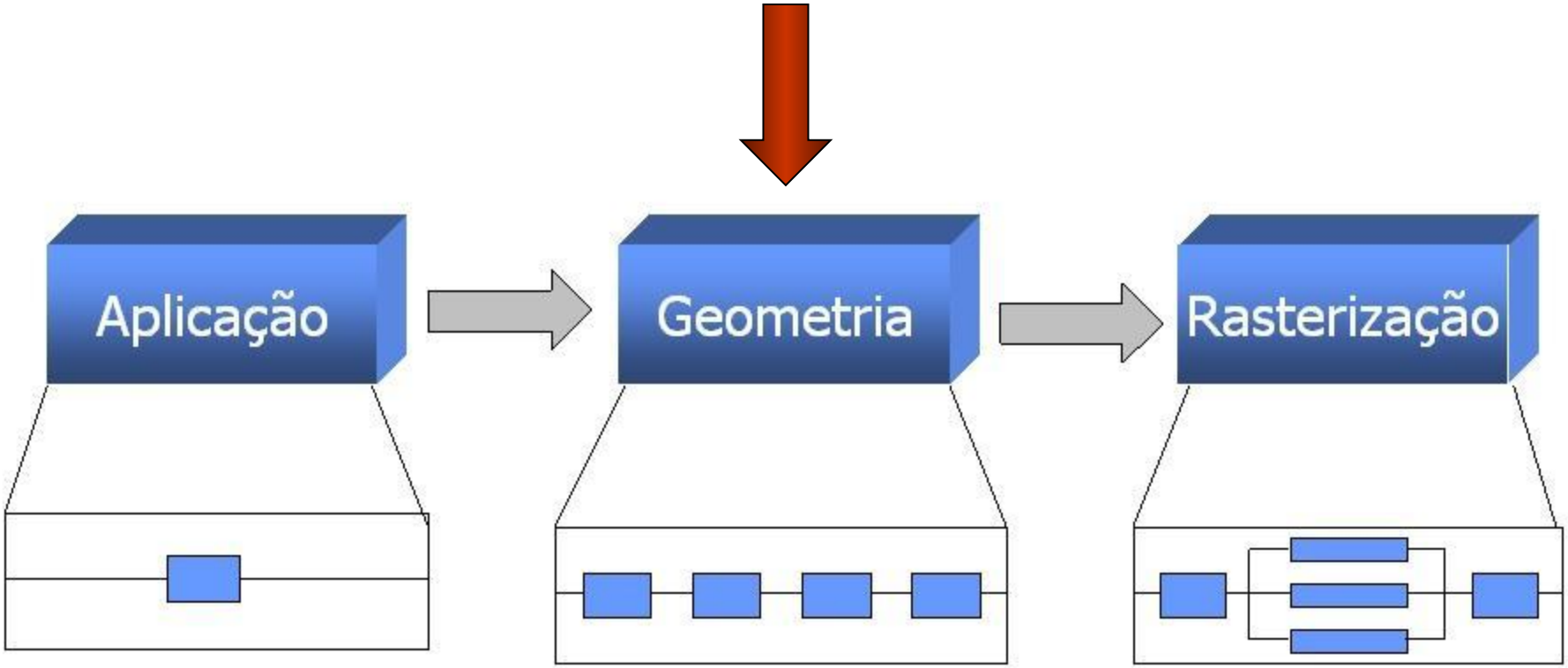
Jogo

Batman: Arkham Knight - Rocksteady

Definições

- **Estágios Funcionais**: estágios que definem a execução de alguma tarefa, mas não **como** esta deve ser realizada.
- **Estágios de Pipeline**: são executados simultâneamente com todos os outros estágios do pipeline.
- Estátios de pipeline podem ser implementados em paralelo para otimizar o processo e dois ou mais estágios funcionais podem ser englobados em um único estágio de pipeline.

Pipeline Gráfico

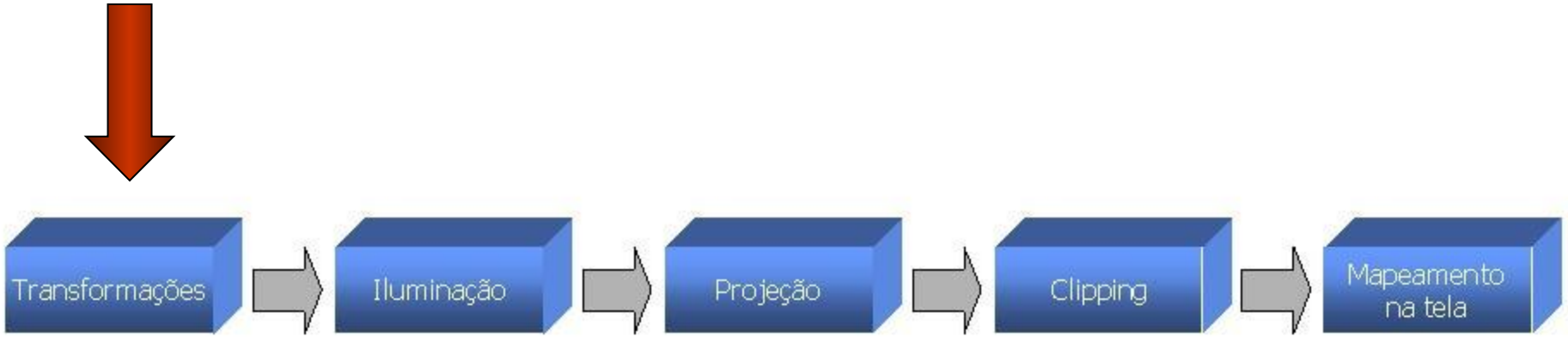


Pipeline Gráfico - Geometria

- Este estágio possui a maioria das operações que podem ser realizadas nos objetos, polígonos a serem exibidos.
- O estágio pode ser dividido em vários estágios **funcionais** que podem ou não ser equivalentes aos estágios de **pipeline**, ou serem implementados em vários estágios de pipeline menores, ou agrupados em estágios de pipeline macros, ou modificar a ordem de execução.

Pipeline Gráfico – Geometria

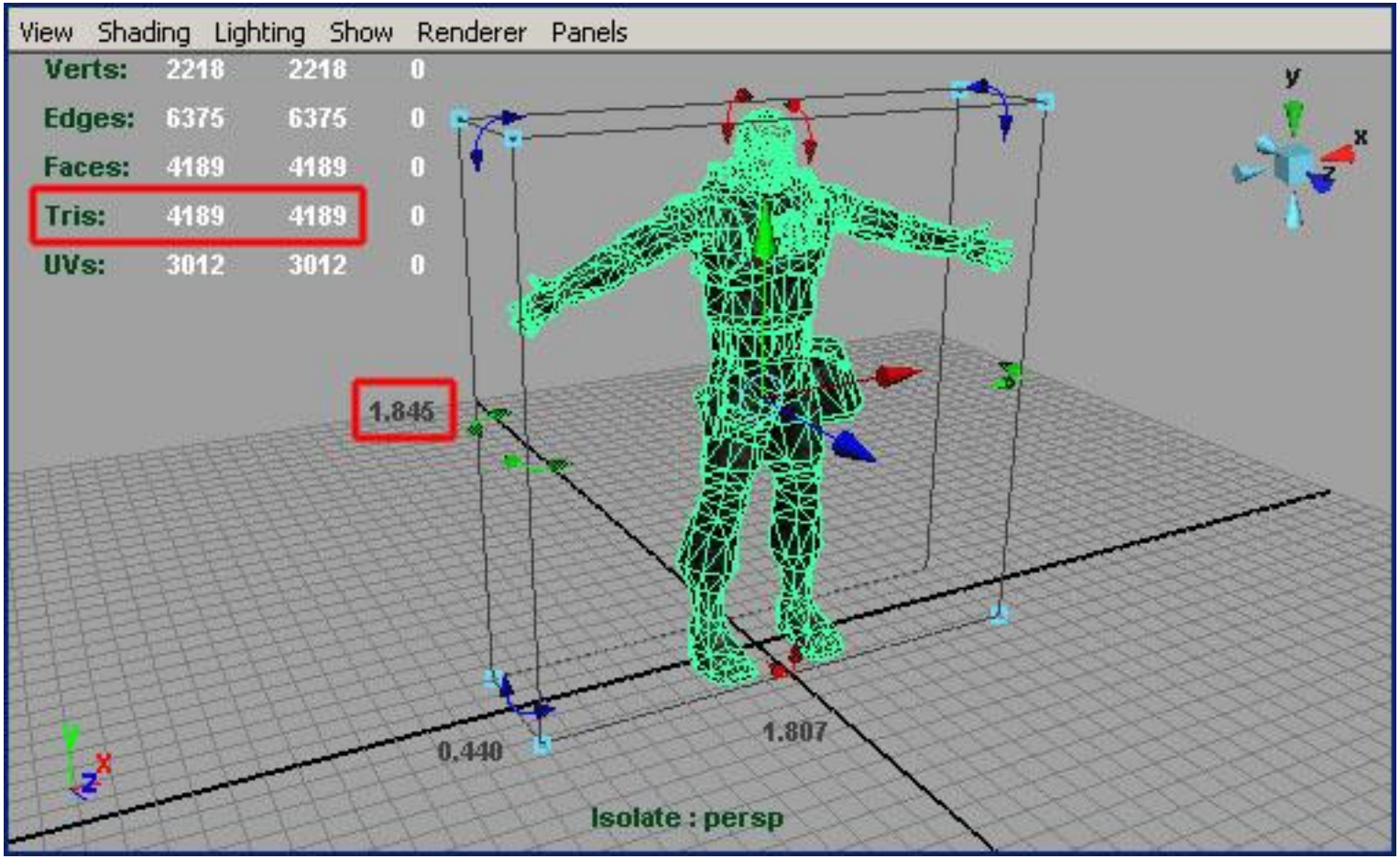
Pipeline Gráfico – Geometria



Estágio de Transformação

- **Transformação** envolve mudar o sistema de coordenada dos objetos (chamdo de *Sistema de Coordenadas do Modelo* SCM – *Model Coordinate* System) a serem renderizados para sistemas mais adequados.
- O que sofre a transformação nos objetos são seus *vértices* e os *vetores normais* associados a eles. Esta transformação recebe o nome de ***Model Transform***
- Após a transformação ser aplicada, diz-se que os objetos estão no *Sistema de Coordenadas do Mundo* SSM (*World Coordinate System*).

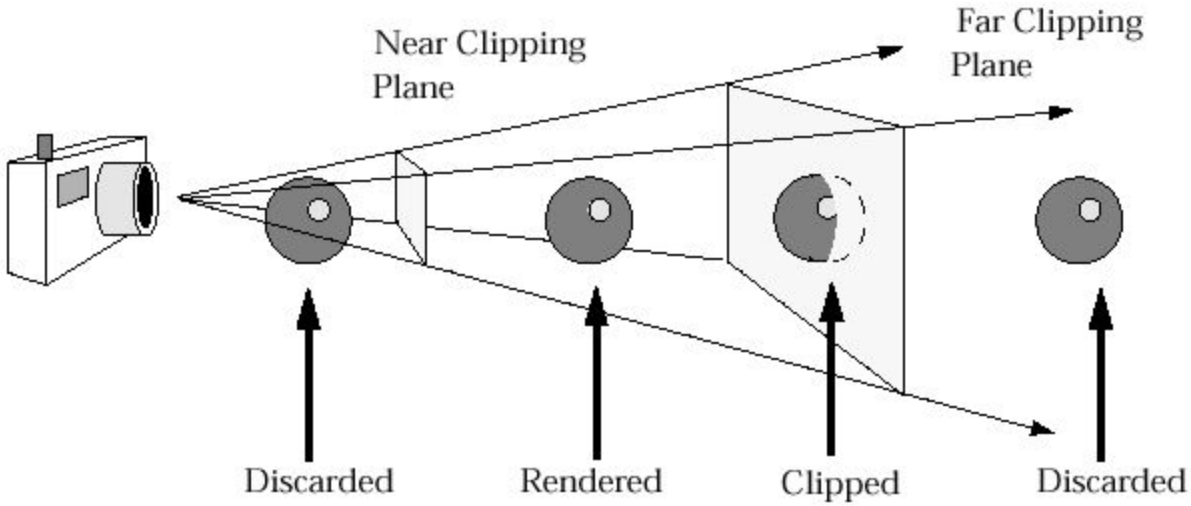
Sistemas de Coordenadas do Modelo



Estágio de Transformação

- Os modelos de computação gráfica definem um objeto especial em uma cena denominado *Câmera*, que serve para capturar a região da cena vista no dispositivo de saída.
- Esta região visível para a câmera se chama *Volume de Visualização*
- A Câmera também possui coordenadas e orientação no SSM que irá sofrer uma transformação.

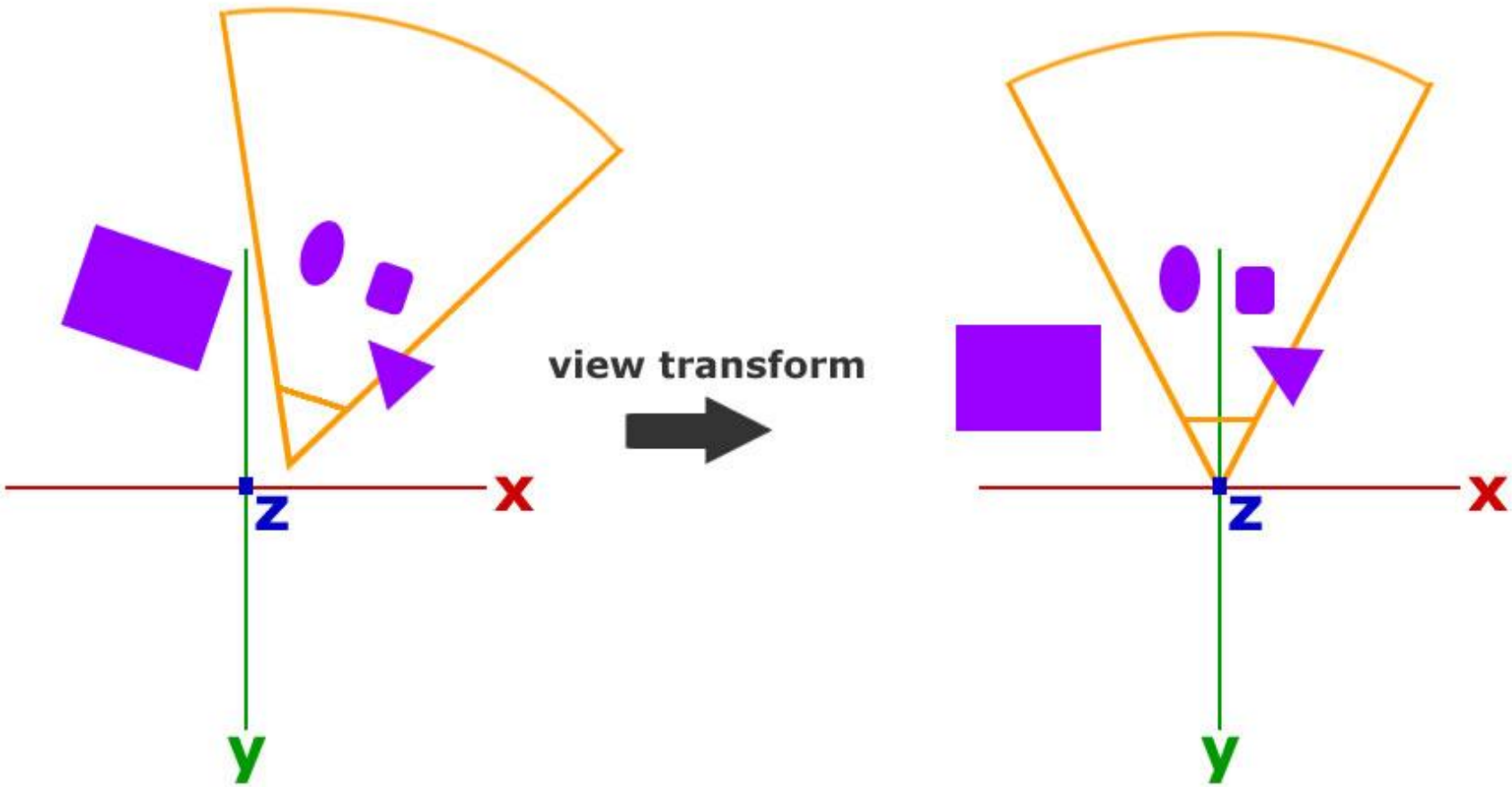
Câmera com o volume de vizualização



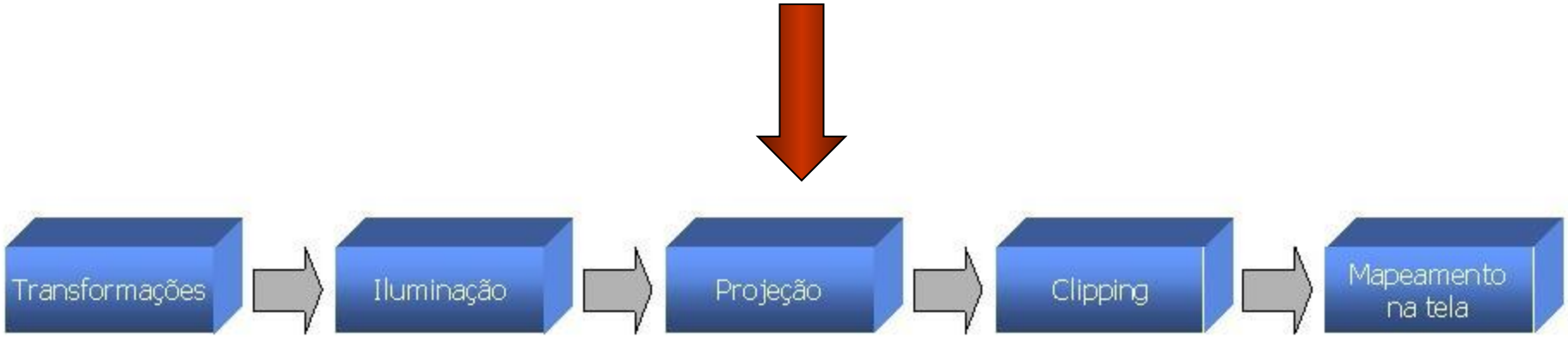
Estágios de Transformação

- Para facilitar outras operações no estágio de Geometria, as coordenadas dos objetos e da câmera são transformados com a ***View Transform***.
- Seu objetivo é localizar a câmera em uma posição especial (em geral a origem) apontando para uma dada direção (eixo Z, no sentido positivo ou negativo).
- Em geral as duas tranformações são concatenadas em uma única chamada ***Model View Transform***.

Model View Transform



Pipeline Gráfico – Geometria



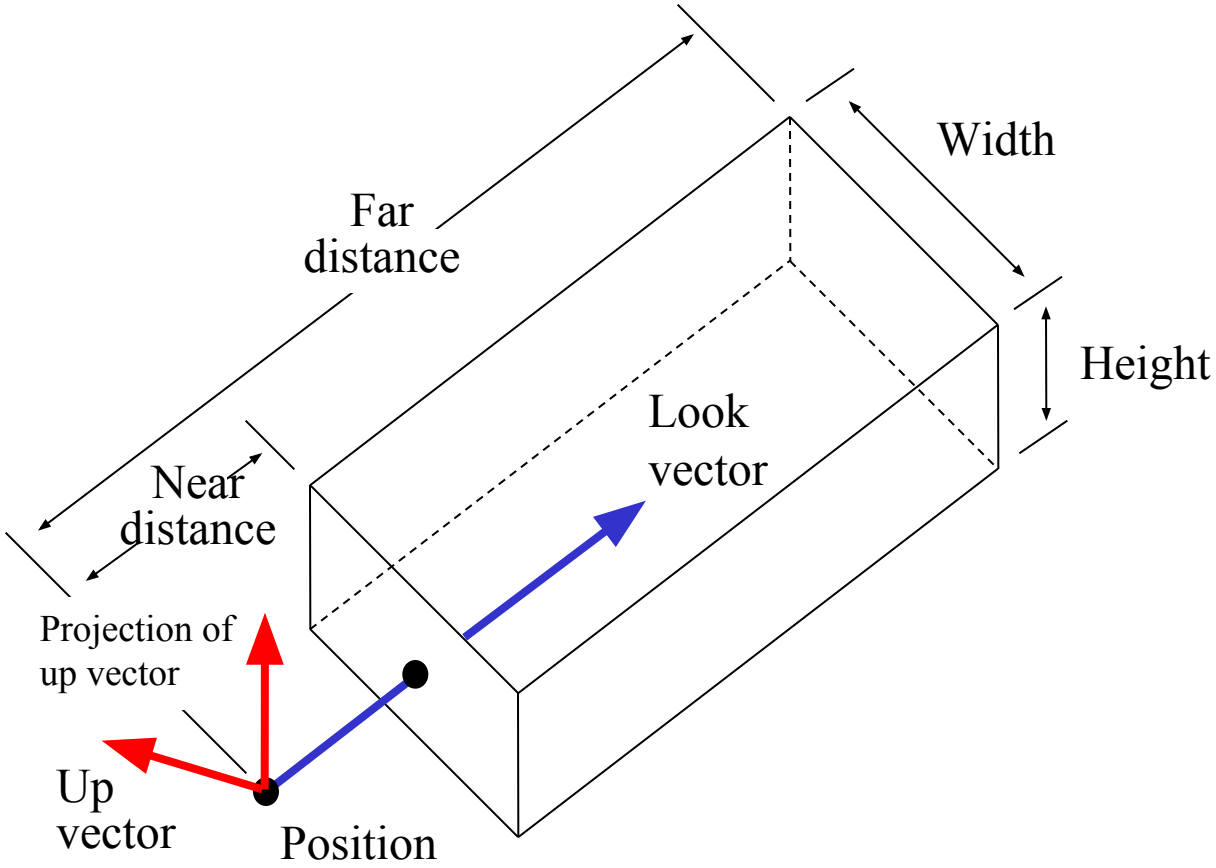
Estágio de Projeção

- **Projeção** transforma o volume de visualização da câmera em um cubo de volume unitário com extremidades (-1,-1,-1) e (1,1,1).
- Os dois tipos de projeção mais utilizados são: **ortográfica** (ou **paralela**) e **perspectiva**.

Estágio de Projeção

- **Projeção Ortográfica** torna o volume de vizualização em um cubo retangular. Sua principal característica é manter os objetos do mesmo tamanho, independente da sua distância em relação à camera

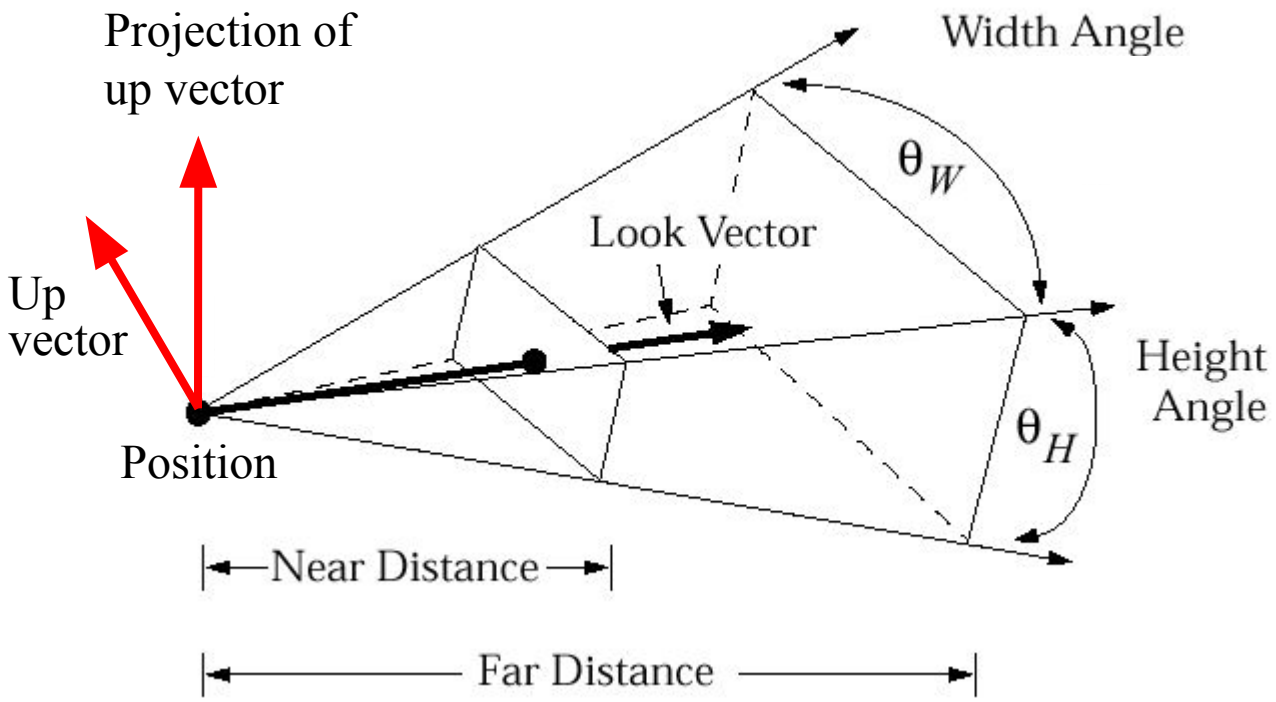
Projeção Ortográfica



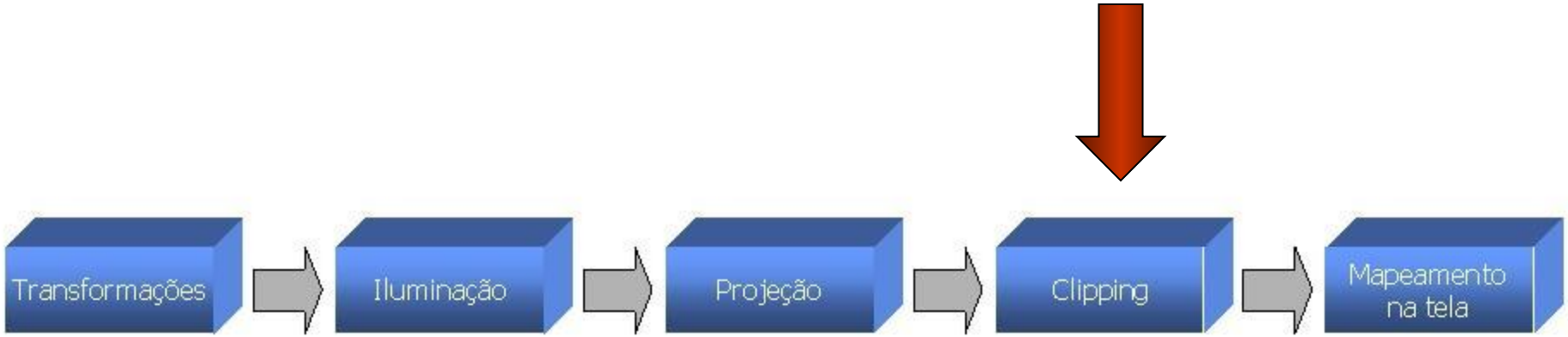
Estágio de Projeção

- **Projeção em Perspectiva** exibe os objetos com tamanhos maiores ou menores, conforme estiverm mais próximos ou mais distantes da câmera. Volume de visualização se torna um frustum.

Projeção em Perspectiva

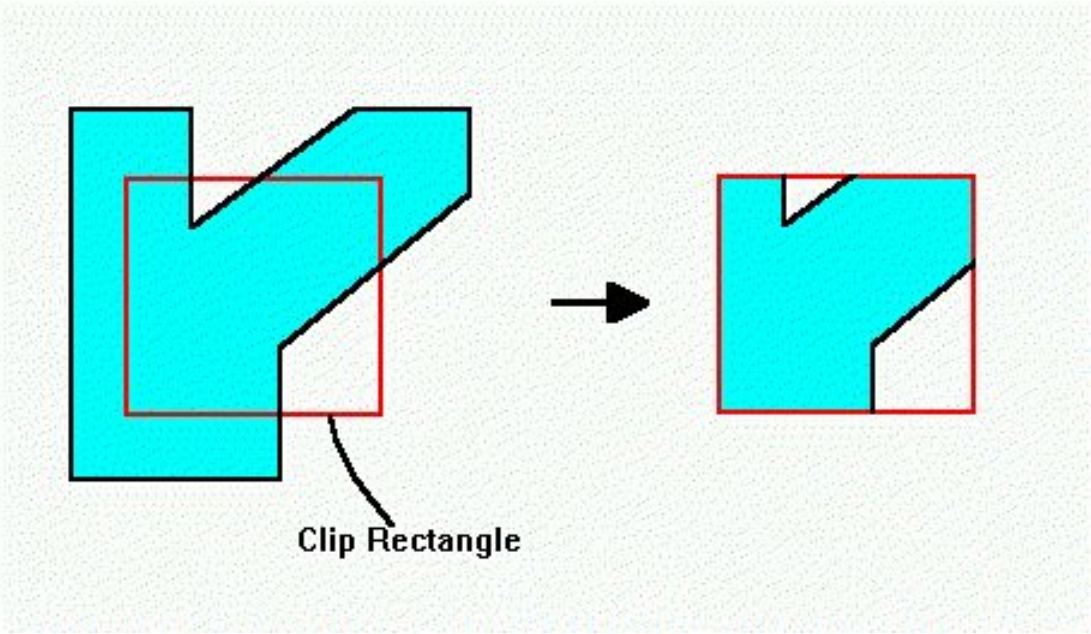


Pipeline Gráfico – Geometria

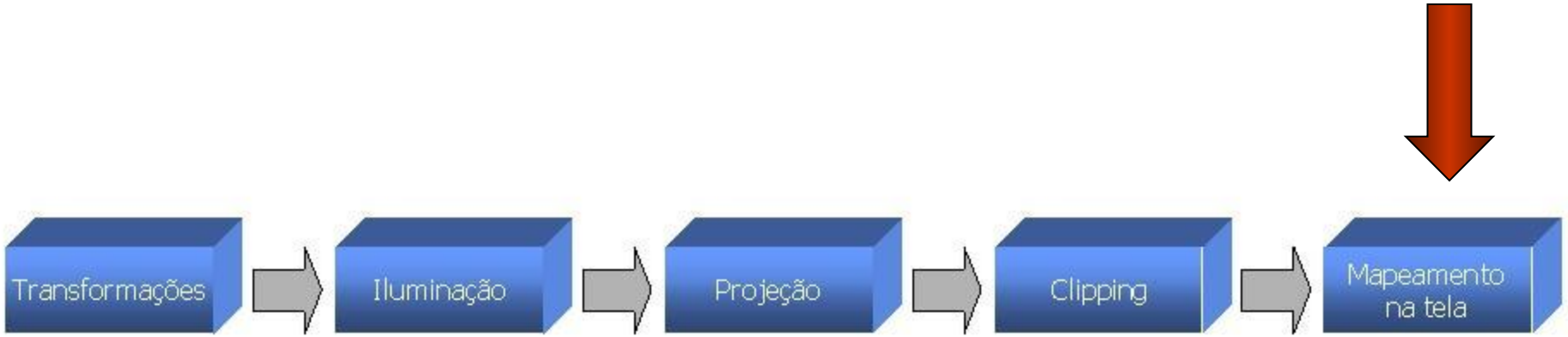


Estágio de *Clipping*

- ***Clipping*** é o processo que determina quais objetos estão parcial ou completamente dentro do volume de visualização.



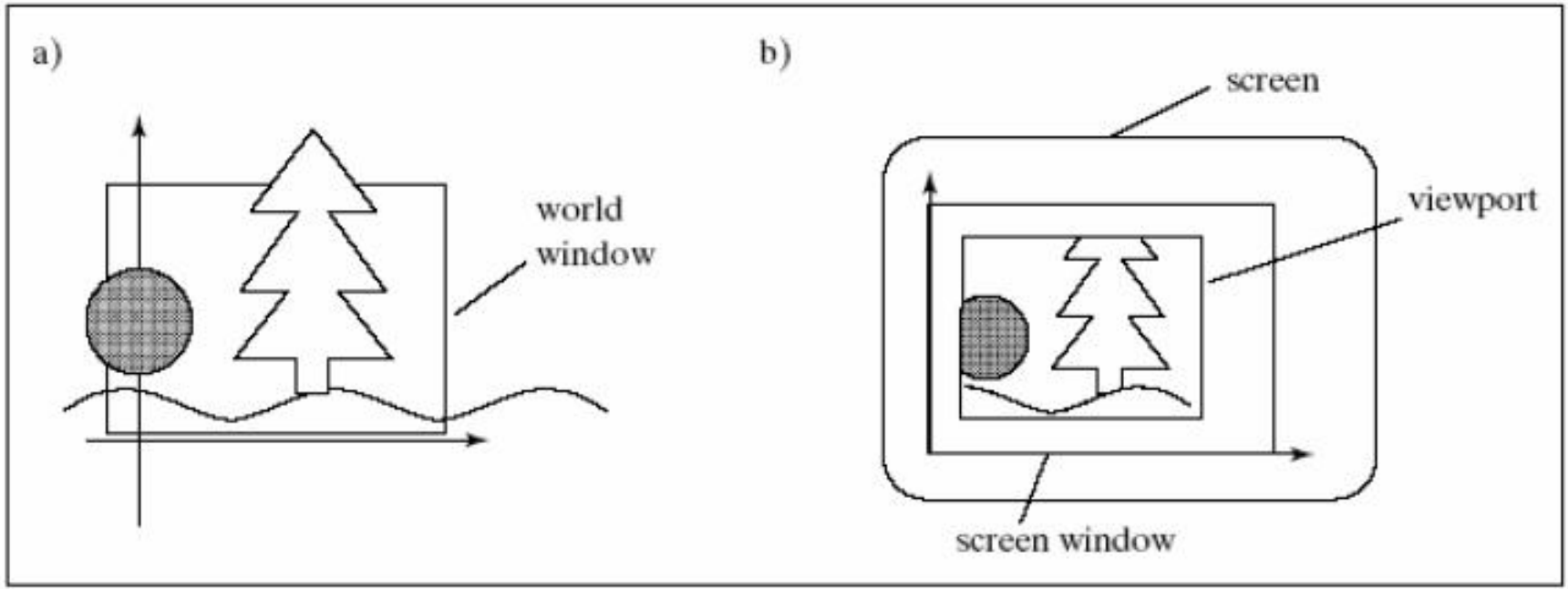
Pipeline Gráfico – Geometria



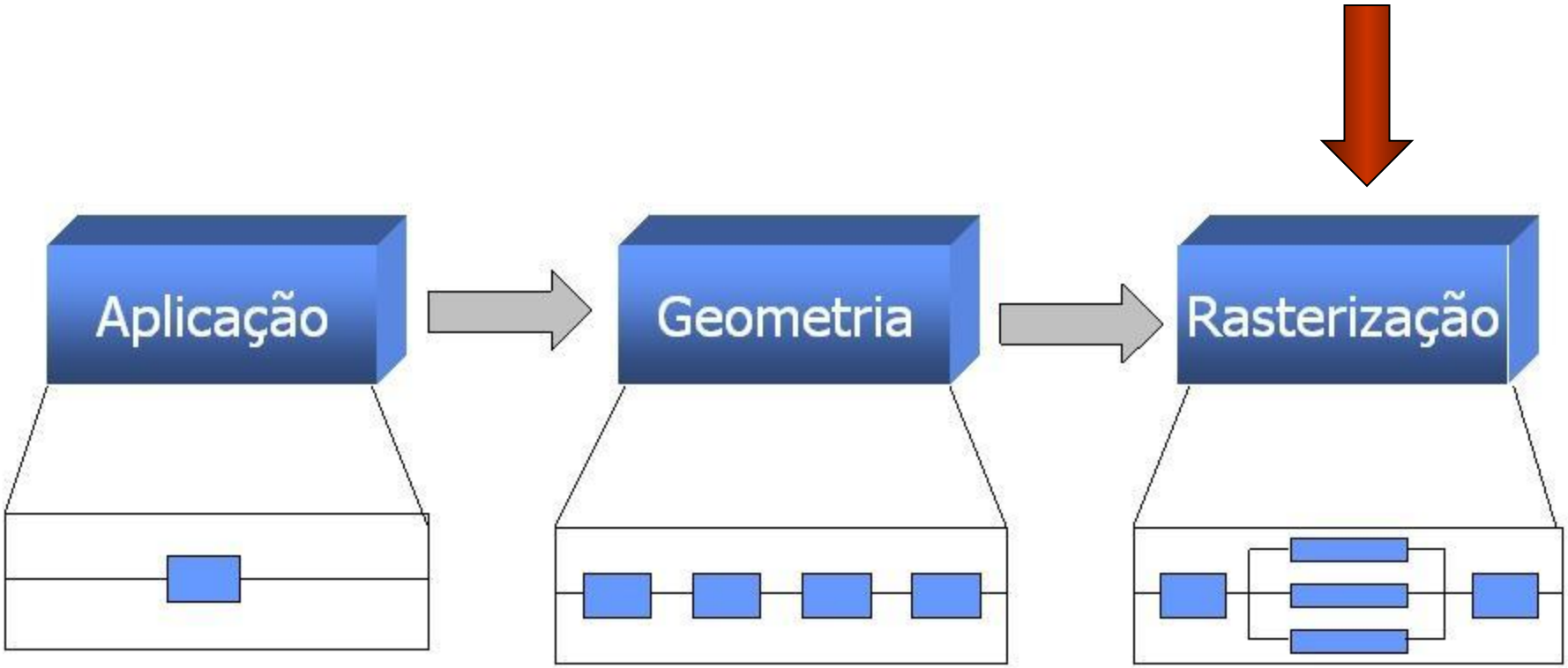
Estágio de Mapeamento na Tela

- **Mapeamento na Tela** é transformar e escalonar as coordenadas X e Y dos objetos para as coordenadas X’ Y’ de uma região, em geral retangular, no dispositivo de saída.
- A coordenada Z não sofre alteração. Esta coordenada é armazenada em uma região especial da memória chamada *Z-Buffer* desde o estágio de projeção.

Estágio de Mapeamento na Tela



Pipeline Gráfico



Pipeline Gráfico – Rasterização

- **Rasterização** pega todas as informações dos objetos já projetados, iluminados, e que passaram pelos outros estágios e transforma estas em **pixeis** para serem renderizados.
- Em geral esta etapa trabalha com dois *buffers* (*front* e *back*) para que o observador veja apenas a imagem final e não o processo da renderização.
- Determinação de visibilidade, ou sobreposição, dos objetos também é feita neste estágio. Se dois objetos possuem regiões em um mesmo pixel, o *rasterizador* irá exibir somente aquele que tiver a coordenada Z mais próxima da câmera.

Tópicos da Apresentação

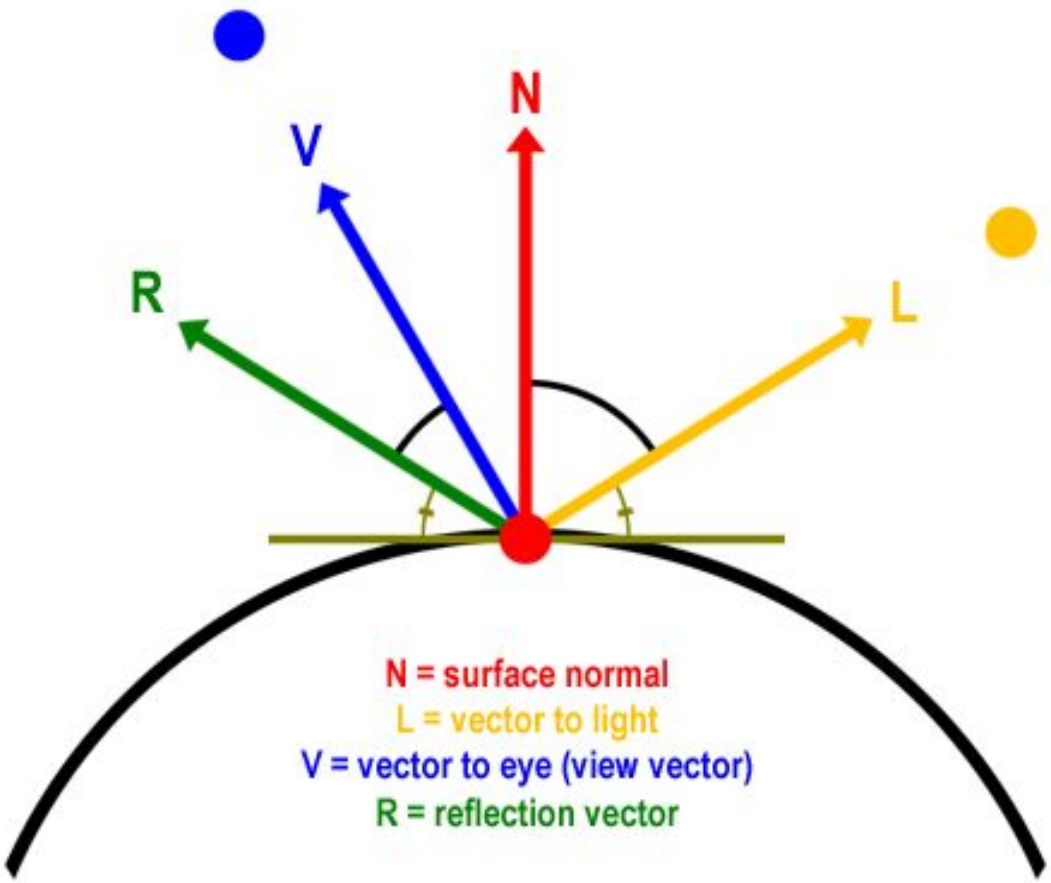
- Contextualização do tutorial.
- Pipeline Gráfico.
- **Modelos de Iluminação.**
- Shaders, Vertex, Geometry, Tesselation e Fragment Shaders.

Definições

- ***Lighting*** refere-se ao processo de interação entre as propriedades dos objetos e as luzes presentes em uma cena.

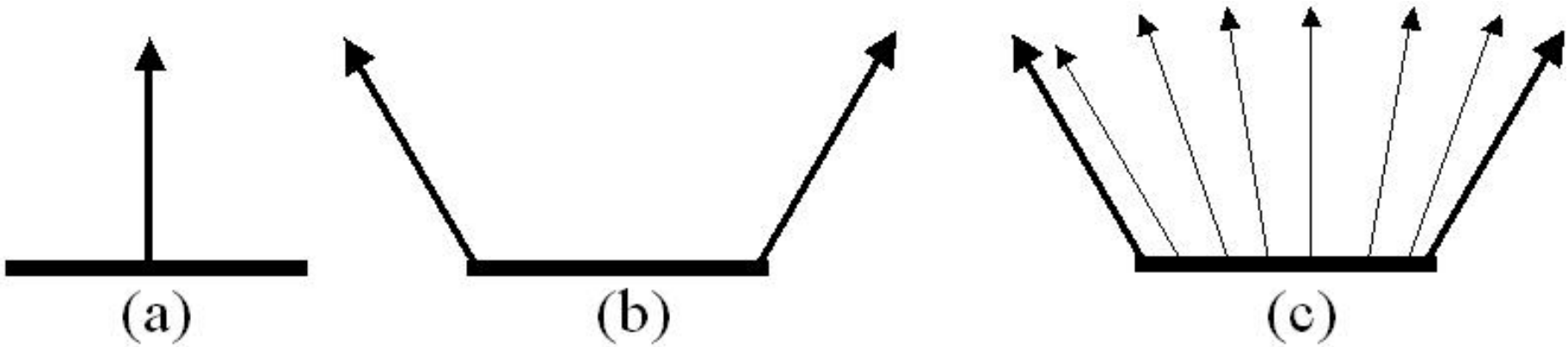
- ***Shading*** é o processo de se executar *lighting* para determinar as cores finais dos pixeis.

Iluminação em um ponto



Principais Algoritmos de Iluminação

Influência do Vetor Normal da Iluminação



Flat **Gourad** **Phong**

Tópicos da Apresentação

- Contextualização do tutorial.
- Pipeline Gráfico.
- Modelos de Iluminação.
- **Shaders, Vertex, Geometry, Tesselation e Fragment Shaders.**

Shader Histórico

- **Paradigma Procedural** é substituir a utilização de dados pré-computados em algumas operações por algoritmos que os calcula em tempo real.
- Na área gráfica o paradigma procedural foi incentivado principalmente pela aparição da especificação do **Renderman**.
- Esta especificação trouxe o conceito de ***Shader Language***, uma linguagem que permite implementar descrições complexas de como as operações de *lighting* e *shading* podem ocorrer.

Shaders

- ***Shaders*** são programas escritos em liguagens apropriadas que permitem substituir certas operações existentes no *pipeline* gráfico implementadas em hardware.
- A especificação Renderman introduziu vários tipos de shaders: *light source shaders, volume shaders, transformation shaders, surface shaders* entre outros.
- Do ponto de vista de aplicações em jogos, os hardwares disponíveis costumam implementar três tipos de shaders: ***Vertex Shader, Geometry, Tesselation*** e **Fragment *Shader***

Vertex Shaders

- ***Vertex Shader***: são shaders que podem alterar os parâmetros dos vértices de um polígono, como a sua cor, seu vetor normal, suas coordenadas de textura.
- Pode-se conseguir alguns efeitos interessantes com vertex shaders: sombras volumétricas, motion blur, silhuetas, deformações procedurais como cabelos, água.

Vertex Shaders Exemplos

Simulação de Água

Pelos

Motion Blur

Geometry Shader

- **Geometry Shaders** podem adicionar ou remover vértices de uma malha quando esta já está no pipeline gráfico. Pode ser utilizado para adicionar conteúdo geométrico proceduralmente.
- Função do GS é permitir os mesmos recursos que VS mas criando-os em tempo real na GPU, reduzindo o tempo de processamento.

Geometry Shaders Exemplos

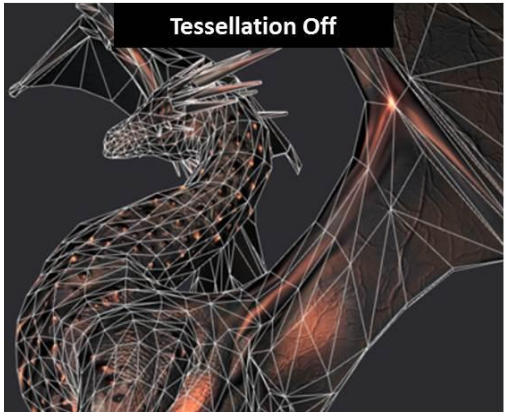
Displacement Mapping

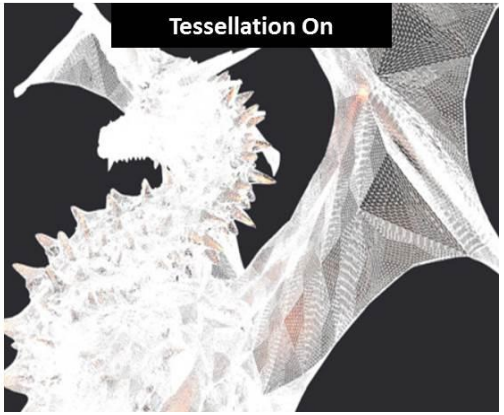
Volumetric Shadows

Tesselation Shader

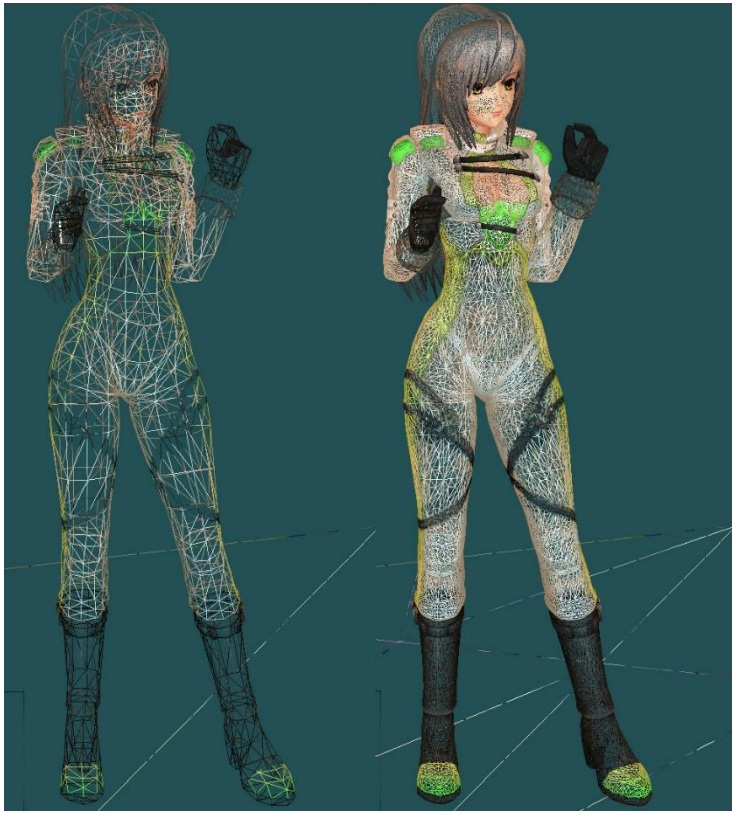
- **Tesselation Shaders** envolvem subdividir uma malha e calcular novos valores para os vértices (posição, normal, cor, coordenada de texturas).

Tesselation Shaders
Exemplos





Subdivisão



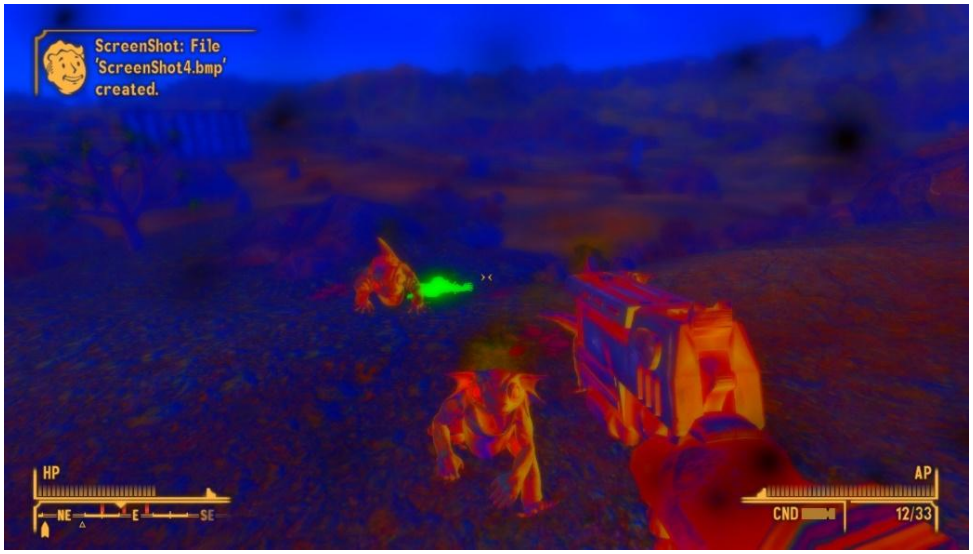
Fragment Shader

- **Fragment Shader**, também chamado de **Pixel Shader**, são operações que podem atuar modificando valores associados a pixeis individuais.
- Estas operações podem atuar em valores individuais, interpolados ou em mapas de texturas para produzirem a cor do pixel.
- Qualquer operação que se realiza nos pixels após a geração do Frame é chamado de Fragment ou Pixel Shader.
- Uma forma bastante utilizada atualmente de pixel shader é substituir o cálculo do vertor normal dos vertex por um mapa de normais.

Fragment Shaders Exemplos

Glow

Edge Detection



Referências

- Livro **Essential Mathematics for Games and Interactive Applications**
- Site, **Essential Math for Game Programmers**

http://www.essentialmath.com/tutorial.htm

- Indispensável: **Vetores, Matrizes e Transformações Afins.**
- Livro **Unity Shaders and Effects Cookbook**
- http://acko.net/files/fullfrontal/fullfrontal/webglmath/online.html
- https://www.youtube.com/watch?v=GNO_CYUjMK8

Referências

- GLSL Wiki Book
  http://en.wikibooks.org/wiki/GLSL_Programming/Unity

- Cg Wiki Book
  http://en.wikibooks.org/wiki/Cg_Programming/Unity

- Cg Tutorial

http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter01.html

- Palestra oficial

https://www.youtube.com/watch?v=epixwRw80MM

Referências

- Videos tutoriais em SurfaceShaders

http://www.creativetd.com/?page_id=617

- Blog de Adrian Courrèges

http://www.adriancourreges.com/blog/

- Blog de Fabien Sanglards

http://fabiensanglard.net/

Games from Scratch

- **Handmade Hero**

https://www.youtube.com/user/handmadeheroarchive/featured

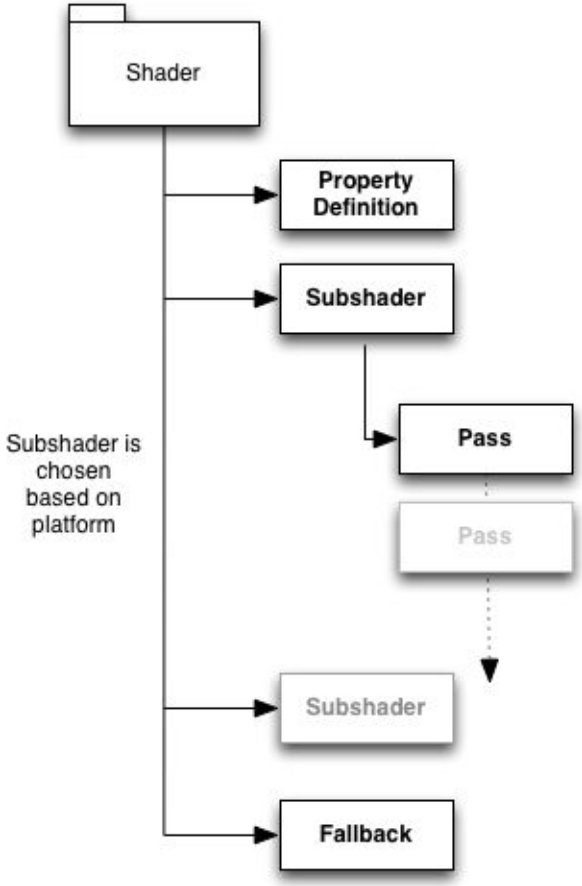
- **Ginger Games**

https://www.youtube.com/user/GingerGames

Shaders no Unity

- Escritos na linguagem **ShaderLab** baseado na linguagem ***Cg*** da ***nVidia***.
- 4 Tipos:
    1. ***Fixed Function Shaders***: para hardwares antigos e efeitos mais simples.
    2. ***Surface Shaders*:** melhor opção para efeitos de interação com lighting e shadow
    3. ***Vertex e Fragment Shaders:*** mais flexíveis mas mais difíceis de escrever.
    4. Em Windows é possível escrever direto em **DX11**.

Esquema geral de Shaders



Properties

- Equivalente a variáveis em uma classe:
- **`Properties { _Name ("DisplayedName", type) = value }`**
- Tipos:
    - Color
    - 2D (Texture potência de 2)
    - Rect (Textura, qualquer dimensão)
    - Cube (Cube Map Texture)
    - Range (min, max)
    - Float
    - Vector

Referências

Mythbusters CPU vs GPU

DirectX 11 Rendering

Properties Exemplo

```
Properties {
    _WaveScale ("Wave scale", Range (0.02,0.15)) = 0.07
    _ReflDistort ("Reflection distort", Range (0,1.5)) = 0.5
    _RefrDistort ("Refraction distort", Range (0,1.5)) = 0.4
    _RefrColor ("Refraction color", Color) = (.34, .85, .92, 1)
    _ReflectionTex ("Environment Reflection", 2D) = "" {}
    _RefractionTex ("Environment Refraction", 2D) = "" {}
    _Fresnel ("Fresnel (A) ", 2D) = "" {}
    _BumpMap ("Bumpmap (RGB) ", 2D) = "" {}
}
```

Subshaders

- Equivalente a funções numa classe:
- Cada ***`Pass`*** corresponde a uma interação em um Loop para modificar a aparência da imagem.
- https://www.youtube.com/watch?v=kEqLGKTQUVQ

Subshaders Fixed – Texture

- Texturas são setadas com o comando:

```
SetTexture[_Name] {
   código;
}
```

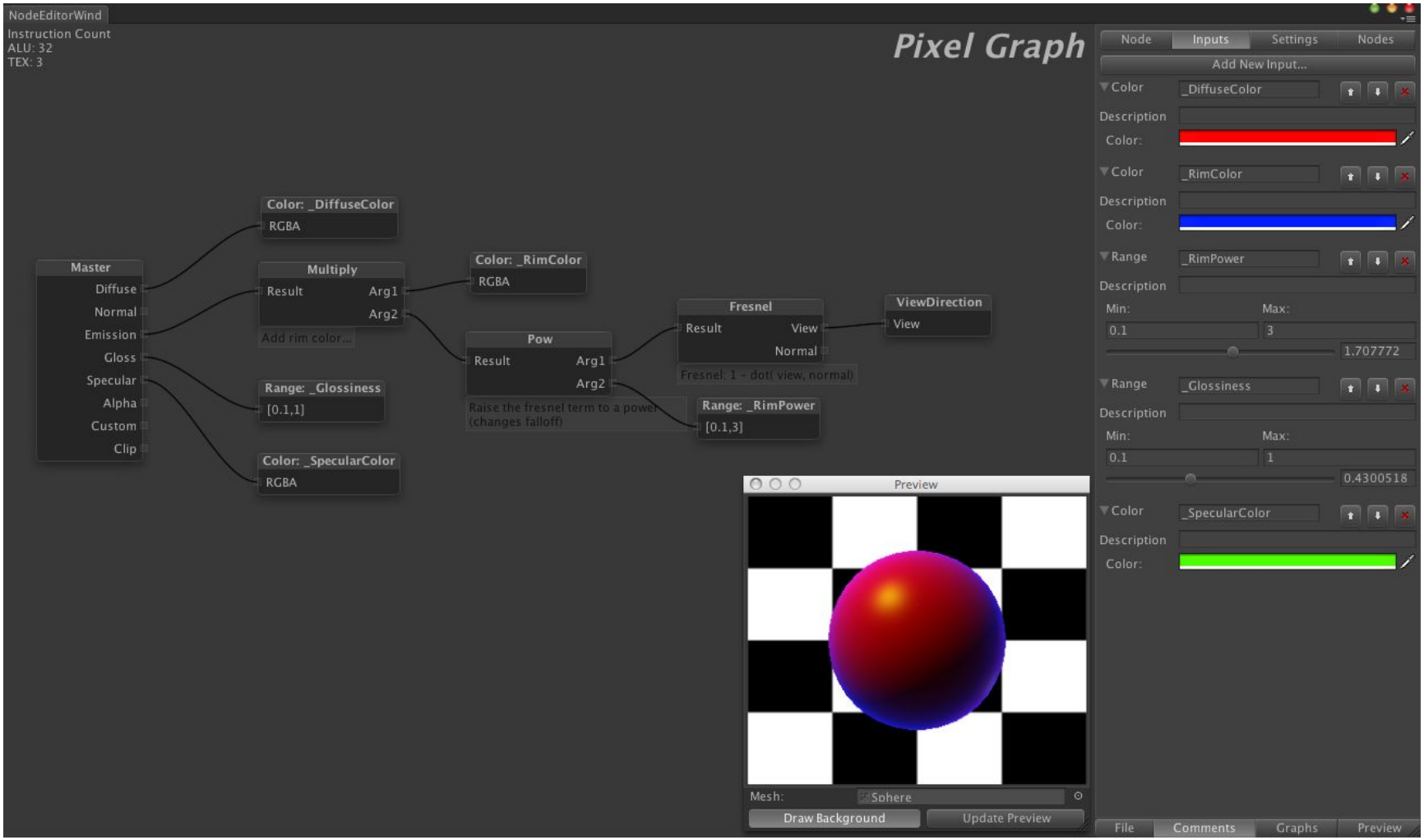
- Sendo que `código` é uma combinação de ações entre o resultado da passagens (`Pass`) de `SetTexture` anterior com o `SetTexture` atual.

Cg/ShaderLab Tipos de Dados

- `vec2 xyCoordinate;`
- `float4 color;`
- `float3 multipliedColor = color.rgb * coordinate.x;`
- Tipos dos parâmetros de entrada e retorno são definidos como `struct`, sendo que o de saída costuma ser sempre o mesmo, `SurfaceOutput`.

```
struct Input {
   float4 color;
   float2 uv_MainText;
};
```

Fixed Function Shaders
Strumpy Editor



Shader Forge